

# Kursory

- W celu wykonania rozkazu SQL system tworzy pewien obszar roboczy nazywany przestrzenią kontekstu
- W przestrzeni tej przechowywane są informacje o przetwarzanym programie w PL/SQL
- PL/SQL pozwala nazwać przestrzeń kontekstu i odwoływać się do zawartych w niej danych za pomocą mechanizmu nazywanego kursorem

# Kursory

- Jeśli zapytanie zwraca wiele wierszy, to możliwe jest utworzenie kursora, który będzie umożliwiał dostęp do pojedynczych wierszy ze zwracanej listy
- Kursory jawne są jedyną metodą umożliwiającą w programie PL/SQL odczyt zbioru rekordów

## **Sposób użycia kursora:**

- zadeklarowanie w sekcji DECLARE
- otwarcie, czyli wykonanie zapytania związanego z kursorem – odczytane z bazy danych rekordy trafiają do pamięci
- pobieranie kolejnych rekordów
- zamknięcie kursora – zwolnienie obszaru pamięci przydzielonego do kursora

# Deklaracja kursora

## Postać deklaracji

```
DECLARE  
CURSOR nazwa_kursora[(lista_parametrów)] IS  
{ zapytanie | RETURN typ_rekordowy }  
[FOR UPDATE [OF lista_atrybutów]];
```

nazwa\_parametru typ [{ := | **DEFAULT** } wartość domyślna]

# Deklaracja kursora

## Przykład

```
DECLARE  
CURSOR samochod IS  
SELECT * FROM samochody;  
CURSOR auto(id NUMBER) IS  
SELECT marka, model, koszt_dnia  
FROM samochody  
WHERE id_sam = id;  
CURSOR klient IS  
RETURN klienci%ROWTYPE;
```

# Otwieranie kursora

## Postać polecenia

**OPEN** nazwa\_kursora[(lista parametrów aktualnych)];

```
DECLARE  
CURSOR samochod IS  
SELECT * FROM SAMOCHODY;  
CURSOR modele(p_marka VARCHAR2) IS  
SELECT MODEL, POJ_SIL, NR_REJ  
FROM SAMOCHODY WHERE MARKA= p_marka;  
BEGIN  
OPEN samochod;  
OPEN modele('OPEL');
```

...

# Pobieranie rekordów z kursora

## Postać polecenia

```
FETCH nazwa_kursora INTO {lista zmiennych  
prostych | zmienna rekordowa};
```

```
DECLARE  
CURSOR klient IS SELECT * FROM KLIENCI;  
id KLIENCI.ID_KLI%TYPE;  
nazw KLIENCI.NAZWISKO%TYPE;  
im KLIENCI.IMIE%TYPE;  
kli KLIENCI%ROWTYPE;  
BEGIN  
OPEN klient;  
FETCH klient INTO id, nazw, im;  
FETCH klient INTO kli;
```

...

# Zamykanie kursora

## Postać polecenia

```
CLOSE nazwa_kursora;
```

```
DECLARE  
CURSOR klient IS SELECT * FROM KLIENCI;  
kli KLIENCI%ROWTYPE;  
BEGIN  
OPEN klient;  
FETCH klient INTO kli;  
...  
CLOSE klient;  
...
```

# Atrybuty kursora

<b>%FOUND</b>	TRUE jeśli ostatnie pobranie rekordu zakończyło się powodzeniem, w przeciwnym przypadku FALSE, NULL przed pierwszym pobraniem
<b>%NOTFOUND</b>	TRUE jeśli ostatnie pobranie rekordu zakończyło się niepowodzeniem, w przeciwnym przypadku FALSE, NULL przed pierwszym pobraniem
<b>%ISOPEN</b>	TRUE jeśli cursor jest otwarty, w przeciwnym przypadku FALSE
<b>%ROWCOUNT</b>	liczba pobranych z kursora rekordów, przed pierwszym pobraniem równy 0



# Atrybuty kursora

```
DECLARE  
CURSOR cur_klient IS SELECT * FROM KLIENCI;  
v_klient KLIENCI%ROWTYPE;  
BEGIN  
IF NOT cur_klient%ISOPEN THEN  
OPEN cur_klient;  
END IF;  
LOOP  
FETCH cur_klient INTO v_klient;  
EXIT WHEN cur_klient%NOTFOUND;  
dbms_output.put_line(to_char(cur_klient%ROWCOUNT)  
|| v_klient.nazwisko);  
END LOOP;  
CLOSE cur_klient;  
END;
```

# Pętla FOR z kursorem

Postać polecenia:

```
FOR licznik IN nazwa_kursora LOOP  
sekwencja poleceń  
END LOOP;
```

- Pętla wykona się tyle razy, ile rekordów odczyta kursor
- Brak konieczności otwierania i zamykania kursora
- Zmienna licznik jest zmienną rekordową, nie należy jej deklarować
- Odwołania do atrybutów tabeli udostępnianej przez kursor możliwe są przy użyciu notacji kropkowej  
**licznik.nazwa\_atrybutu**

# Użycie pętli FOR z kursorem

```
DECLARE  
CURSOR cur_samochod(p_id_sam NUMBER) IS  
SELECT Id_sam, id_kli, data_wyp, data_zwr  
FROM wypozyczenia WHERE id_sam = p_id_sam;  
BEGIN  
FOR v_cur IN cur_samochod(100) LOOP  
  dbms_output.put_line(to_char(cur_samochod%ROWCOUNT)  
  || ' ' || v_cur.id_kli || ' ' || v_cur.id_sam  
  || ' ' || v_cur.data_wyp || ' ' || v_cur.data_zwr);  
END LOOP;  
END;
```

# Pętla FOR z podzapytaniem

Postać polecenia:

```
FOR licznik IN (zapytanie) LOOP  
sekwencja poleceń  
END LOOP;
```

- Pętla wykona się tyle razy, ile rekordów odczyta zapytanie
- Zmienna po której jest pętla jest zmienną rekordową o strukturze rekordu zapytania - nie należy jej deklarować
- Odwołania do atrybutów zapytania możliwe są przy użyciu notacji kropkowej

licznik.nazwa\_atrybutu

# Pętla FOR z podzapytaniem

Przykład:

```
DECLARE
```

```
v_marka samochody.marka%TYPE := '&nazwa_marka';
```

```
BEGIN
```

```
dbms_output.put_line('Modele samochodu marki: ' ||  
v_marka);
```

```
FOR v_mod IN (SELECT model, nr_rej, poj_sil FROM  
samochody
```

```
WHERE marka = v_marka) LOOP
```

```
dbms_output.put_line(v_mod.model || ' ' || v_mod.nr_rej || '  
v_mod.poj_sil);
```

```
END LOOP;
```

```
END;
```

# Klauzula WHERE CURRENT OF

- Umożliwia modyfikację (polecenie UPDATE) albo usunięcie (polecenie DELETE) bieżącego rekordu kursora
- Kursor musi zostać zadeklarowany z klauzulą FOR UPDATE
- Otwarcie kursora zakłada blokady na rekordach odczytanych przez zapytanie kursora

# Klauzula WHERE CURRENT OF

- Jeśli w deklaracji kursora umieszczono klauzulę **FOR UPDATE**, rekordy, odczytane z bazy danych przez kursor, zostają zablokowane i przygotowane do ewentualnej modyfikacji, czy usunięcia
- Jeśli bieżący rekord kursora ma zostać zmodyfikowany, wówczas operację tą realizuje się, używając standardowego polecenia **UPDATE** do tabeli, z której kursor odczytał rekord
- Rekord modyfikowany musi być bieżącym rekordem kursora co zapewnia użycie klauzuli **WHERE CURRENT OF <nazwa kursora>**
- Analogicznie wygląda sytuacja, gdy chcemy usunąć z tabeli rekord, będący bieżącym rekordem kursora – stosujemy wówczas polecenie **DELETE** z klauzulą **WHERE CURRENT OF <nazwa kursora>**

# Klauzula WHERE CURRENT OF

```
DECLARE  
CURSOR cur_samochody(p_id_sam NUMBER) IS  
SELECT marka, koszt_dnia  
FROM samochody WHERE id_sam = p_id_sam  
FOR UPDATE;  
zmiana NUMBER(4,1);  
BEGIN  
FOR marki IN cur_samochody(100) LOOP  
IF marki.marka = 'FORD' THEN zmiana := 1.5;  
ELSE zmiana := 1.1; END IF;  
UPDATE samochody SET koszt_dnia = koszt_dnia * zmiana  
WHERE CURRENT OF cur_samochody;  
END LOOP;  
END;
```



# Kursor niejawny

- Tworzony automatycznie dla poleceń **INSERT, UPDATE, DELETE i SELECT INTO** w programie PL/SQL
- Otwierany bezpośrednio przed wykonaniem polecenia
- Zamykany zaraz po wykonaniu polecenia
- **Nazwa kursora: SQL**

# Atrybuty kursora niejawnego

<b>%FOUND</b>	TRUE jeśli polecenie odczytało lub zmodyfikowało chociaż jeden rekord
<b>%NOTFOUND</b>	TRUE jeśli polecenie nie odczytało lub nie zmodyfikowało żadnego rekordu
<b>%ISOPEN</b>	Zawsze FALSE
<b>%ROWCOUNT</b>	Liczba rekordów odczytanych lub zmodyfikowanych przez polecenie

# Kursor niejawny

```
BEGIN  
DELETE FROM klienci WHERE kod_pocz is null;  
IF SQL%FOUND THEN  
  dbms_output.put_line('Rekordy usuniete: ' ||  
    to_char(SQL%ROWCOUNT));  
ELSE  
  dbms_output.put_line('Brak rekordów do usunięcia!');  
END IF;  
INSERT INTO klienci(id_kli, nazwisko)  
AS SELECT id_kli + 10, 'Iksinski ' || nazwisko FROM  
  klienci;  
  dbms_output.put_line('Dodanych rekordów: ' ||  
    to_char(SQL%ROWCOUNT));  
END;
```

**Zdefiniuj kursor, dzięki któremu będzie można wyświetlić  
trzy najdroższe samochody w wypożyczalni - wykorzystaj  
atrybut kursora %ROWCOUNT**

```
DECLARE  
CURSOR cur_sam IS SELECT marka, model  
FROM samochody ORDER BY koszt_dnia desc;  
modele samochody.model%TYPE;  
marki samochody.marka%TYPE;  
BEGIN  
OPEN cur_sam;  
LOOP  
FETCH cur_sam INTO modele, marki;  
EXIT WHEN cur_sam%NOTFOUND OR cur_sam%ROWCOUNT>3;  
dbms_output.put_line(to_char (cur_sam%ROWCOUNT)  
|| '. ' || marki || ' ' || modele);  
END LOOP;  
CLOSE cur_sam;  
END;
```

# Obsługa wyjątków

- Wyjątek – błąd lub ostrzeżenie, wygenerowane w czasie działania programu

## Rodzaje wyjątków:

- **predefiniowane** – zgłaszane automatycznie przez system, zdefiniowane dla błędów z katalogu Oracle
  - **użytkownika** – zadeklarowane w sekcji DECLARE, zgłaszane przez użytkownika poleceniem RAISE
- Po wystąpieniu wyjątku sterowanie przechodzi do sekcji obsługi wyjątków (klauszula EXCEPTION)

# Sekcja EXCEPTION

- Sekwencja poleceń\_k, która zostaje wykonana, gdy w bloku wystąpi konkretny rodzaj wyjątku\_k
- Nieobsłużony wyjątek przerywa działanie programu
- Opcjonalna sekcja OTHERS – obsługuje wszystkie niewymienione wyjątki

**DECLARE**

...

**BEGIN**

...

**EXCEPTION**

**WHEN** nazwa wyjątku\_1 **THEN**  
sekwencja poleceń\_1

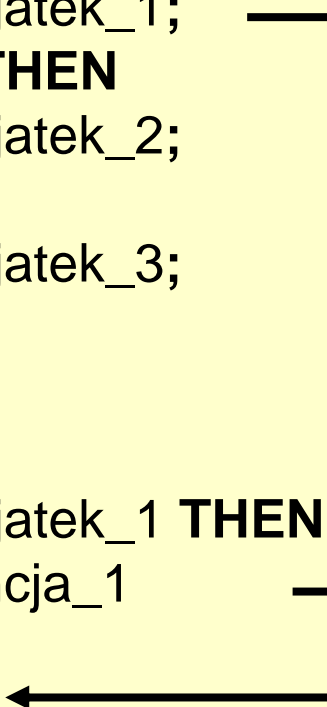
**WHEN** nazwa wyjątku\_2 **THEN**  
sekwencja poleceń\_2

...

**WHEN OTHERS THEN**  
sekwencja poleceń\_k  
**END;**

# Propagacja wyjątków

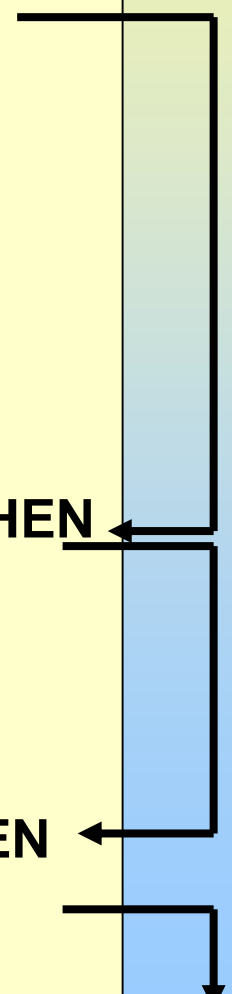
```
BEGIN
  BEGIN
    IF x=1 THEN
      RAISE wyjatek_1;
    ELSIF x=2 THEN
      RAISE wyjatek_2;
    ELSE
      RAISE wyjatek_3;
    END IF;
  ...
  EXCEPTION
    WHEN wyjatek_1 THEN
      sekwencja_1
    END;
  ...
  EXCEPTION
    WHEN wyjatek_2 THEN
      sekwencja_2
    END;
```



wyjatek **wyjatek\_1** jest obsługowany lokalnie w bloku wystąpienia wyjątku (zostaje wykonana **sekwencja\_1**), sterowanie przechodzi do kolejnej instrukcji w bloku nadrzędnym

# Propagacja wyjątków

```
BEGIN
  BEGIN
    IF x=1 THEN
      RAISE wyjatek_1;
    ELSIF x=2 THEN
      RAISE wyjatek_2;
    ELSE
      RAISE wyjatek_3;
    END IF;
  ...
  EXCEPTION
    WHEN wyjatek_1 THEN
      sekwencja_1
    END;
  ...
  EXCEPTION
    WHEN wyjatek_2 THEN
      sekwencja_2
    END;
END;
```



nie znaleziono klauzuli obsługi **wyjatek\_2** w bloku wystąpienia wyjątku, **wyjatek\_2** jest obsłużony w bloku nadrzędnym (zostaje wykonana **sekwencja\_2**)  
-program zostaje zakończony



# Propagacja wyjątków

```
BEGIN
  BEGIN
    IF x=1 THEN
      RAISE wyjatek_1;
    ELSIF x=2 THEN
      RAISE wyjatek_2;
    ELSE
      RAISE wyjatek_3;
    END IF;
  ...
  EXCEPTION
    WHEN wyjatek_1 THEN
      sekwencja_1
    END;
  ...
  EXCEPTION
    WHEN wyjatek_2 THEN
      sekwencja_2
    END;
```

nie znaleziono klauzuli obsługi **wyjatku\_3** ani w bloku wystąpienia wyjątku ani w bloku nadrzędnym, program zostaje przerwany z komunikatem „wystąpił nieobsłużony wyjątek”

BŁĄD

# Predefiniowane wyjątki

Nazwa wyjątku	Nr błędu	opis wyjątku
<b>CASE_NOT_FOUND</b>	ORA-06592	nie znaleziono pasującej klauzuli WHEN dla CASE
<b>CURSOR_ALREADY_OPEN</b>	ORA-06511	próba otwarcia już otwartego kursora
<b>DUP_VAL_ON_INDEX</b>	ORA-00001	powielenie wartości w atrybucie kluczem podst. lub unikalnym
<b>INVALID_CURSOR</b>	ORA-01001	wykonanie zabronionej operacji na kursorze
<b>NO_DATA_FOUND</b>	ORA-01403	polecenie SELECT INTO nie zwróciło żadnego rekordu
<b>TOO_MANY_ROWS</b>	ORA-01422	polecenie SELECT INTO zwróciło więcej niż jeden rekord
<b>VALUE_ERROR</b>	ORA-06502	błąd wykonania op. arytmetycznej, konwersji lub rozmiaru typu
<b>ZERO_DIVIDE</b>	ORA-01476	dzielenie przez zero

# Predefiniowane wyjątki - przykład

**DECLARE**

v\_model samochody.model%TYPE;

**BEGIN**

**SELECT** model **INTO** v\_model

**FROM** samochody **WHERE** marka = 'FORD';

dbms\_output.put\_line(' Dla marki FORD mamy model: '||v\_model);

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

dbms\_output.put\_line('Brak modeli dla marki FORD');

**WHEN TOO\_MANY\_ROWS THEN**

dbms\_output.put\_line('Więcej niż jeden model dla marki FORD');

**WHEN OTHERS THEN**

dbms\_output.put\_line('Mamy jakiś inny błąd');

**END;**

# Wyjątki użytkownika

- Deklaracja:

**DECLARE**

nazwa\_wyjątku EXCEPTION;

- Wywołanie:

**RAISE** nazwa\_wyjątku;

- Poleceniem **RAISE** można również wywołać wyjątki predefiniowane, np. **RAISE TOO\_MANY\_ROWS**

# Wyjątki użytkownika

**DECLARE**

**tanie\_samochody EXCEPTION;**

**drogie\_samochody EXCEPTION;**

**id samochody.id\_sam%TYPE := :id\_samochodu;**

**CURSOR cur\_sam(p\_id\_sam NUMBER) IS**

**SELECT koszt\_dnia FROM samochody WHERE id\_sam = p\_id\_sam FOR UPDATE;**

**BEGIN**

**FOR v\_sam IN cur\_sam(id) LOOP**

**BEGIN**

**IF v\_sam.koszt\_dnia < 200 THEN RAISE tanie\_samochody;**

**ELSE RAISE drogie\_samochody; END IF;**

**EXCEPTION**

**WHEN tanie\_samochody THEN UPDATE samochody SET koszt\_dnia =  
koszt\_dnia \* 1.2 WHERE CURRENT OF cur\_sam;**

**WHEN drogie\_samochody THEN UPDATE samochody SET koszt\_dnia =  
koszt\_dnia / 1.2 WHERE CURRENT OF cur\_sam;**

**END;**

**END LOOP;**

**END;**

# RAISE\_APPLICATION\_ERROR

- Polecenie **RAISE\_APPLICATION\_ERROR** przerywa działanie programu z wypisaniem na konsoli komunikatu o wystąpieniu błędu

Użycie:

```
RAISE_APPLICATION_ERROR(numer_błędu, komunikat);
```

- Numer błędu: od –20000 do –20999

# RAISE\_APPLICATION\_ERROR

**DECLARE**

v\_model samochody\_model%TYPE := '&nazwa\_modelu';

v\_id\_sam samochody.id\_sam%TYPE;

**BEGIN**

**SELECT** id\_sam **INTO** v\_id\_sam

**FROM** samochody **WHERE** model = v\_model;

**UPDATE** wypozyczenia **SET** id\_sam = null **WHERE** id\_sam IN  
(**SELECT** id\_sam **FROM** samochody  
**WHERE** id\_sam = v\_id\_sam);

**DELETE** samochody **WHERE** id\_sam = v\_id\_sam;

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**raise\_application\_error**(-20001,'Nie wypożyczono modelu  
samochodu '|| v\_model);

**END;**